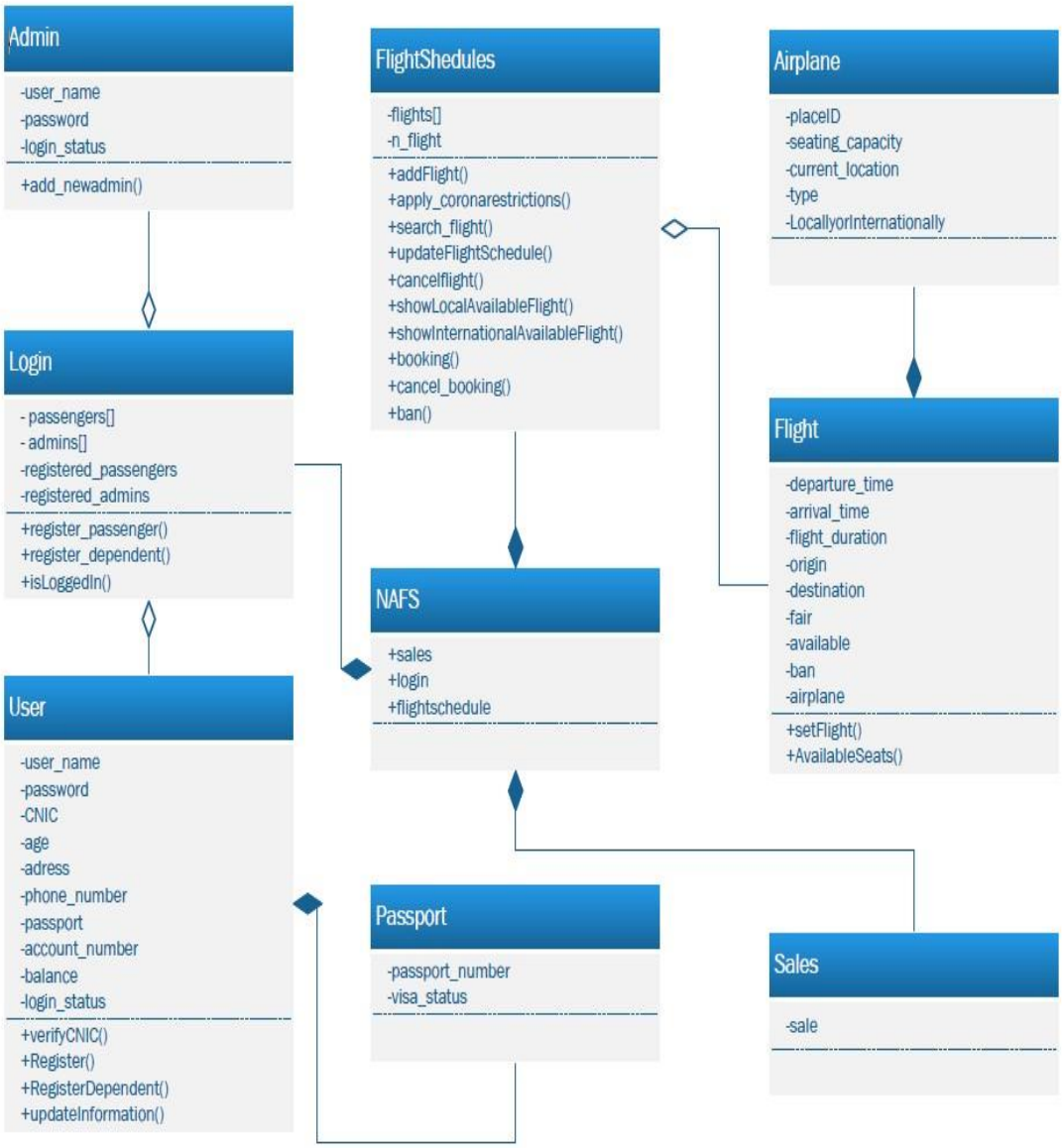
Project Report

Class Diagram

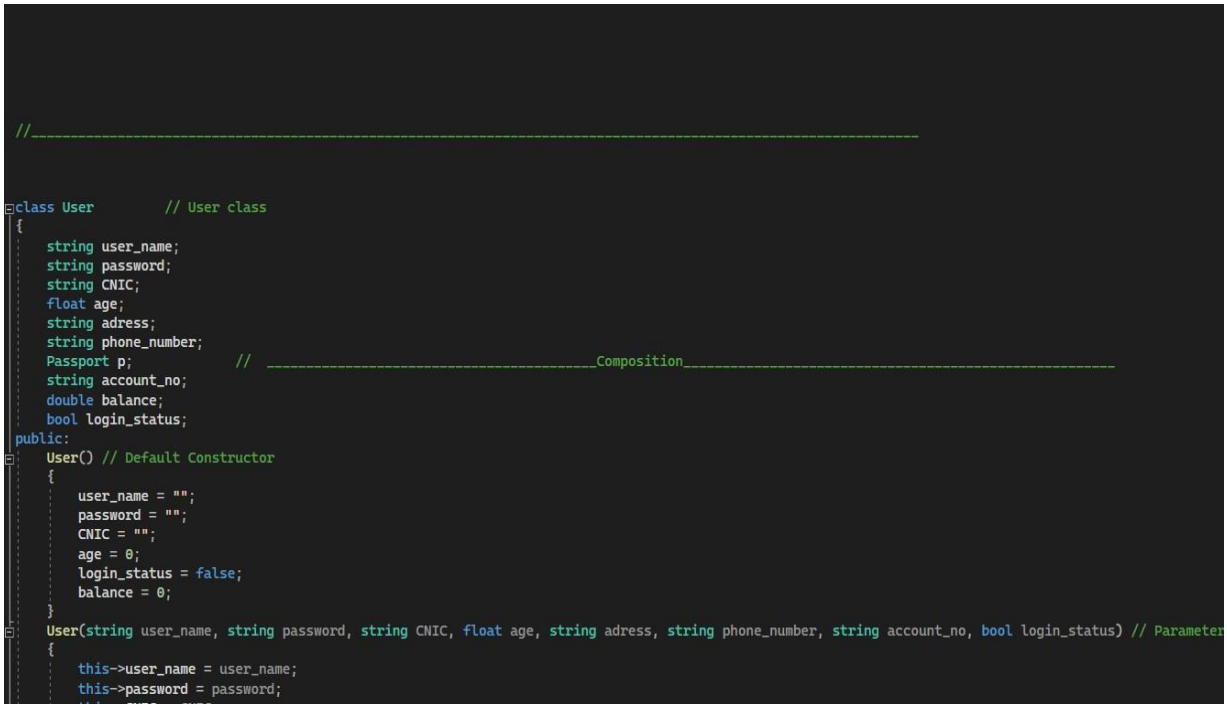


Class Admin

- This class is made for storing admin basic info like name, password and login status.
- The admin can add a new admin by using add_newadmin() function.

Class User

- This class stores user basic details.
- verify_CNIC() function verifies the pattern of entered CNIC.
- Register() function registers the new user.
- RegisterDependent() function registers the dependent of the registered user.
- UpdateInformation() function updates the passenger's information.
- The Passport class is **composed** in this class because if a user will exist he must have a passport or not so that's why passport is composed in it.



```
//_____________________________________________________________________________________________________


class User         // User class
{
    string user_name;
    string password;
    string CNIC;
    float age;
    string adress;
    string phone_number;
    Passport p;            //  _____________________________________Composition_____________________________________
    string account_no;
    double balance;
    bool login_status;
public:
    User() // Default Constructor
    {
        user_name = "";
        password = "";
        CNIC = "";
        age = 0;
        login_status = false;
        balance = 0;
    }
    User(string user_name, string password, string CNIC, float age, string adress, string phone_number, string account_no, bool login_status) // Parameter
    {
        this->user_name = user_name;
        this->password = password;
```

Class Login

- It has 2 arrays, one is passengers and the other is admins that are respectively **aggregated** from User and Admin class because the all user and admins can exist without login and their array consist their informations.

```
class Login  //  Login class
{
    User* passengers;    // (_____aggregation_____)
    Admin* admins;       // (_____aggregation_____)
    int registered_passengers;
    int registered_admins; // Only can be changed when the manager will add any admin detail in "Admins.txt" file
public:
    Login() // Default constructor
    {
        int i = 0;
        string h;
        string name;
        string password;
        string balance;
        string visa;
        ifstream read("Passengers Information.txt");
        while (read.eof() == 0)
        {
            getline(read, h);
            i++;
        }
        read.close();
        registered_passengers = i / 4;
        passengers = new User[registered_passengers];
        ifstream read2("Passengers Information.txt");
        int o= 0;
        for (int i = 0; i < registered_passengers; i++)
        {
            getline(read2, name);
            getline(read2, password);
            getline(read2, balance);
            getline(read2, visa);
            passengers[o].setUsername(name);
```

- register_passenger() adds the passenger in the passenger array.
- register_dependent() adds the dependent passenger in the passenger array.
- IsLoggedIn() function cheaks whether the passenger is logged in or not.

Class Passport

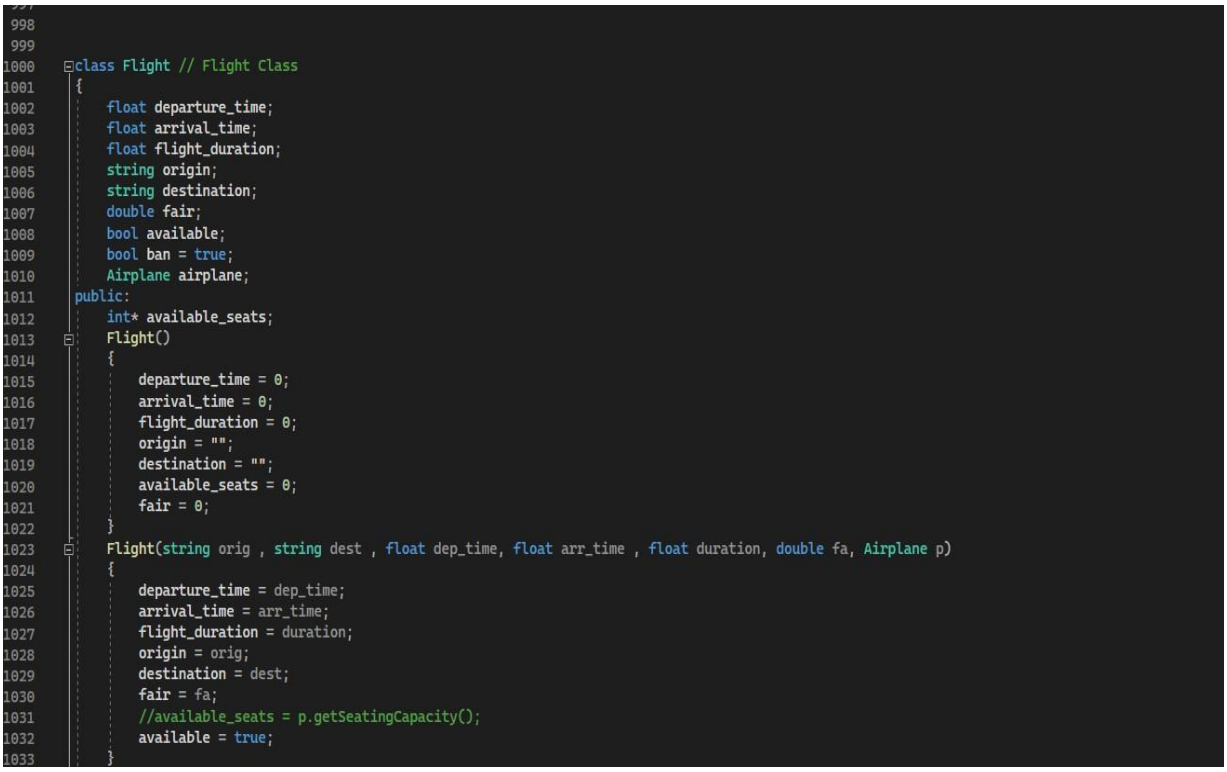
- This class consists of passport number and passport visa status.

Class Airplane

- This class stores airplaneID, plane seating capacity, plane current location, plane type and tell the aiplane is set for local or international flights.

Class Flight

- It stores the basic details of the flight like arrival and departure time, flight duration, origin and destination and fair etc.
- The Airplane class is **composed** in it because if a flight exist it must have an airplane.

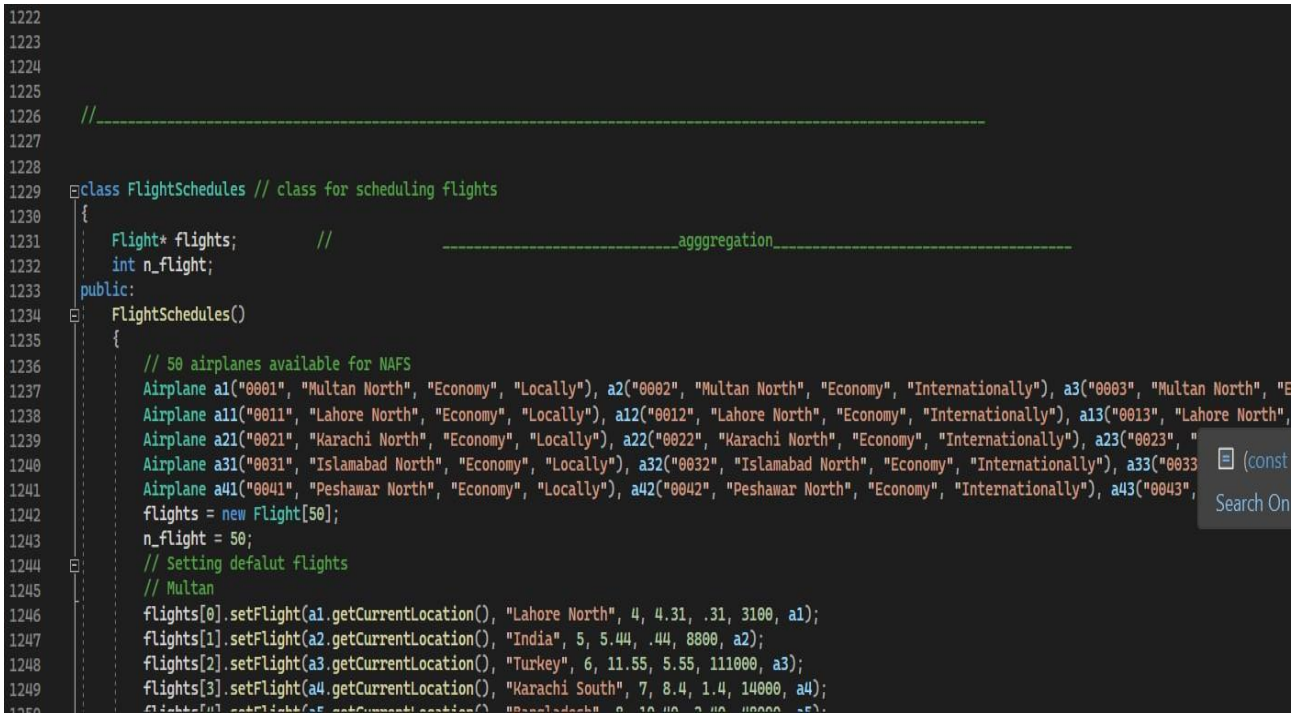


```
class Flight // Flight Class
{
    float departure_time;
    float arrival_time;
    float flight_duration;
    string origin;
    string destination;
    double fair;
    bool available;
    bool ban = true;
    Airplane airplane;
public:
    int* available_seats;
    Flight()
    {
        departure_time = 0;
        arrival_time = 0;
        flight_duration = 0;
        origin = "";
        destination = "";
        available_seats = 0;
        fair = 0;
    }
    Flight(string orig , string dest , float dep_time, float arr_time , float duration, double fa, Airplane p)
    {
        departure_time = dep_time;
        arrival_time = arr_time;
        flight_duration = duration;
        origin = orig;
        destination = dest;
        fair = fa;
        //available_seats = p.getSeatingCapacity();
        available = true;
    }
```

- setFlight() function is for seting the attributes of flights
- AvailableSeats() function tell the available seats in that airplane.

Class FlightShedules

- It has an array of Flights thus Flights are **aggregated** in this class. Because a flight array should be made in this and flight can exist without flight schedule class.



```
1222
1223
1224
1225
1226  //_____________________________________________________________________________________________
1227
1228
1229  class FlightSchedules // class for scheduling flights
1230  {
1231      Flight* flights;          //            _____________________agggregation_____________________
1232      int n_flight;
1233  public:
1234      FlightSchedules()
1235      {
1236          // 50 airplanes available for NAFS
1237          Airplane a1("0001", "Multan North", "Economy", "Locally"), a2("0002", "Multan North", "Economy", "Internationally"), a3("0003", "Multan North", "E
1238          Airplane a11("0011", "Lahore North", "Economy", "Locally"), a12("0012", "Lahore North", "Economy", "Internationally"), a13("0013", "Lahore North",
1239          Airplane a21("0021", "Karachi North", "Economy", "Locally"), a22("0022", "Karachi North", "Economy", "Internationally"), a23("0023", "
1240          Airplane a31("0031", "Islamabad North", "Economy", "Locally"), a32("0032", "Islamabad North", "Economy", "Internationally"), a33("0033
1241          Airplane a41("0041", "Peshawar North", "Economy", "Locally"), a42("0042", "Peshawar North", "Economy", "Internationally"), a43("0043",
1242          flights = new Flight[50];
1243          n_flight = 50;
1244          // Setting defalut flights
1245          // Multan
1246          flights[0].setFlight(a1.getCurrentLocation(), "Lahore North", 4, 4.31, .31, 3100, a1);
1247          flights[1].setFlight(a2.getCurrentLocation(), "India", 5, 5.44, .44, 8800, a2);
1248          flights[2].setFlight(a3.getCurrentLocation(), "Turkey", 6, 11.55, 5.55, 111000, a3);
1249          flights[3].setFlight(a4.getCurrentLocation(), "Karachi South", 7, 8.4, 1.4, 14000, a4);
```

- addFlight() function adds the new flight to the flight schedule if all planes are not booked.
- apply_coronarestrictions() __The admin uses this function to apply restrictions of corona like baninig double seats.
- search_flight() function searches the flights from flights array.
- updateFlightSchedule() function automatically updates the flight according to the given conditions in project.
- cancelflight() function cancel the flight by taking airplane id.
- showLocalAvailableFlights() function shows the all locally flights
- showInternationalAvailableFlights() function shows the all International flights.
- booking() function books the seats and adjust balances.
- cancel_booking() class cancel your booking and give your money back.
- ban() function bans flights to a given country.

Class Sales

- This class checks and balances the money transfers.
- set_Sales() function reads company profit from a file and add booking money in this and updates the file.

struct NAFS

- This structure have Sales, Login and FlightSchedules classes **composed** in it because if NAFS exist then the other classes will exist that are composed or aggregated in each other.
- It is the main structure that controls all things.

```
1770
1771
1772
1773
1774    //_____________________________________________________________________________________________________
1775
1776
1777    struct NAFS
1778    {
1779        Sales sales;
1780        Login l;
1781        FlightSchedules f;
1782    };
1783
1784
1785
1786
1787
1788
1789
1790    //_____________________________________________________________________________________________________
1791
```